(19) Eur päisches Patentamt
European Pat nt Office
Office europé n des brev ts

(11) Publication number: **0 324 308 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **88480070.7**

(51) Int. Cl.[4]: **G06F 9/30**

(22) Date of filing: **08.11.88**

(30) Priority: **11.01.88 US 142585**

(43) Date of publication of application:
19.07.89 Bulletin 89/29

(84) Designated Contracting States:
DE FR GB

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Larsen, Larry Donald**
912 Emory Lane
Raleigh, NC 27609(US)

(74) Representative: **Bonneau, Gérard**
Compagnie IBM France Département de Propriété Industrielle
F-06610 La Gaude(FR)

(54) Method of decoding plural incompatible format instructions.

(57) This invention relates to the architectural design of a data processing system and its method of operation which permits normally incompatible plural format instructions for dissimilar processors to be placed intermixed in the instruction storage of a single machine but to be accurately decoded and executed properly regardless. Instructions in different formats which are normally incompatible, are placed in predefined or segregated areas of the instruction store (2). Instructions are fetched and decoded in an instruction decode memory (5) in a manner which uses portions of both the fetched-from address in the instruction (2) store and the instruction itself. Decoding is thus determined in part by where in the instruction store the instruction resided when fetched, and the specific instruction itself. This approach is compatible with both ordinary processor architecture and with pipelined processor architectures.

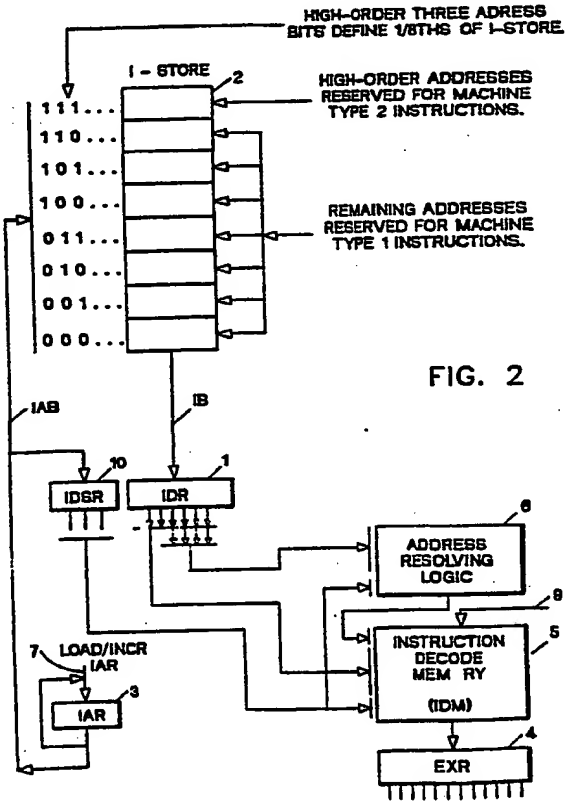


FIG. 2

EP 0 324 308 A2

itself, as is the usual case. This technique and construction allows normally incompatible format or instruction types from two or more machines to be intermixed in the instruction store without requiring extra instruction format bits to identify non-native instructions. It does not require type or mode switch instructions to be inserted in the instruction stream by the programmer. Overall, instructions for a machine of this type consume less instruction space than other approaches.

The invention will be described with reference to a preferred embodiment thereof which is further illustrated in the drawings in which:

Figure 1, consisting of Figures 1A and 1B, illustrates an exemplary three-phase pipelined processor architecture and structure utilizing additional components that facilitate the decoding operation as defined for the invention.

Figure 2 illustrates the specific additional components and method of operation required in the present invention in greater detail.

Figure 3 illustrates two different instruction formats for the same operation as typical examples of two incompatible instruction formats that may be executed to perform the same operation in the processor.

Figure 4 illustrates a specific example of the operation utilizing a first instruction type format for a given instruction in the apparatus of Figures 1 and 2.

Figure 5 illustrates another example of a different format for the same type of execution operation in the structure of Figures 1 and 2.

Figure 6 illustrates the fact that the result of decoding either type of instruction in Figures 4 and 5 produces the same results in the execution register portion of the apparatus as depicted.

A three-phase pipelined data processor architecture is illustrated in the descriptions which follow. In this architecture, an instruction is first fetched on one cycle, decoded in the second cycle and executed in the third cycle by the processor's arithmetic logic unit. Instructions reside in the instruction store 2 in Figure 1 and are fetched into the instruction decoding register 1 for decoding by the instruction decoding logic. The decoding logic, which may consist of some form of memory or may be implemented in a more conventional way using standard logic elements (i.e., ANDS, ORs, etc.), is used to transform fetched instructions to a form appropriate to operate the gates and controls of the processor flow logic in such a way that the desired instruction execute actions tak place. The transformed instruction is then placed in the Execute Register 4, at the end of decode phase. The instruction address register 3 provides the next instruction address to instruction stor 2 over the instruction address bus IAB. Instructions are sequentially fetched and placed in the instruction decode register 1 and portions of the instruction are examined in the address resolving logic 6 which contains logic that defines which field or fields of the instruction are to be used and (perhaps) modified to produce an effective address or direct operand value. In the example shown in Figure 2, decode tables in the instruction decoding memory or logic 5 may be loaded externally on line 9. This may provide the loading of specific instruction decoding rules as will be later described.

The basic ideas presented in this type of structure and architecture apply equally well to the non-pipelined architectural designs for processing machines as well. This will be readily appreciated by those of skill in the art, and while the preferred embodiment discussed utilizes a pipelined architectural design, this is only for convenience in describing the invention, but has no direct bearing upon the present invention.

In Figure 1, the instruction decode register 1, abbreviated IDR, is shown. The IDR is the register into which instructions fetched from the instruction store, called I-store 2, are placed to be decoded. Instructions taken from, or "fetched" from, the I-store 2 are placed in the IDR where they are held for decoding. The instructions are stored in the IDR prior to being decoded in the instruction decoding logic or memory 5. The instruction address register 3, abbreviated IAR in Figure 1, is used to contain the I-store 2 address for the instruction that is to be fetched. The execution register 4, abbreviated EXR, contains decoded instructions which are the execution process codes for the specific machine. Instruction decoding logic elements comprising the instruction decode memory or logic as explained earlier, with its contained logic, and the address resolving logic 6 are utilized for decoding th instructions that reside in the IDR 1. The address resolving logic extracts specific bits from the instruction in the IDR, based upon the instruction type, and then causes an effective address or immediate operand value to be calculated by adding (for example) the contents of some index register (R0 or R4 in Figure 1B).

In the example illustrated in Figure 1, I-store 2 is assumed to be separate from a data store (not shown) and is thus a "harvard" architectural design for the machine. Loading instructions into I-store 2 may be accomplished previously by loading or writing a ROM or RAM chip with the instructions outsid of the host machine. Alternatively, I-store is RAM, and may be written using special instructions in the host processor r pertoire for this purpose. When this is done, the path is via IDB with IAR providing th write address.

erated so that it retains the instruction address or some portion thereof of the instruction currently residing in IDR 1. The IDSR 10 is used to trap or retain the address of the last instruction that was fetched from I-store 2. The IDSR 10 retains a portion of the address of the last instruction fetched and will be clocked with the same signal that causes the IDR 1 to be loaded. The retained portion may be all of the address or less if required. Thus, it will always receive the address of the instruction that is loaded into the IDR 1. In Figure 2, IDSR 10 is assumed to contain only the high order three bits from the IAR 3 because of the arbitrarily assumed partitioning of I-store 2 in the instant example. Only the three high order bits necessary to define which of the eight equal arbitrary segments of I-store or partitions are being addressed.

The contents of IDSR 10 and of IDR 1 are taken together in Figure 2 to provide a look-up address within the IDM 5. Only specific portions of the undecoded instruction in IDR may be needed according to the format of the instruction being fetched. The other portion of the input address to the IDM 5 is the output from the IDSR 10 which identifies the fetched-from address in I-store 2, or in the instant example, at least the range in which the address lies within I-store 2. Decoding of any specific instruction in the IDR 1 thus depends not only on the contents of the IDR 1 but on the region of the I-store 2 from which the instruction was fetched. This portion of the address is provided by the IDSR 10 output.

IDSR 10 also has an output path to the address resolution logic 6. This is necessary to enable the logic to access the appropriate bit fields from the IDR 1 when an operand or effective address must be formed. In general, there will be no compatibility that can be expected between instruction sets of various machines with regard to the placement within the instruction, and the length of, any operand fields.

The effect of using the fetched-from address, or portion of the address, which defines at least a field within the I-store regions, as a part of the access specifications for decoding the specific instruction that is fetched, is to permit instructions in different regions of I-store 2 to be decoded utilizing different rules. Thus, if a portion of I-store 2 is set aside for "type 2programs, as is shown in Figure 2, and if the instruction decode memory 5 is set up in the manner described above, all instructions fetched from the region of I-store 2 that are identified by th three high order addr ss bits 111, will be decoded according to type 2 decode rules contained in IDM 5, and instructions fetched from the type 1 region in the I-store 2 will be decoded in the unique manner required for type 1 instructions.

A more specific example is given as follows:

Let us suppose that a "type 1" format or language instruction set requires 24 bit words and that it is desired to execute also "type 2" format or language instructions that arbitrarily utilize 16 bit words, and that the same machine is to be required to handle both types of instructions. For the example presently given, it will be assumed that each type of original machine for which the instruction words were written has an "add registers" instruction in a repertoire of its commands. It is, of course, understood that the function of each command is to cause the contents of arbitrary registers such as R2 and R3 to be added together and the sum placed in an arbitrary register such as R3. It will be assumed further that the instruction formats like those shown in Figure 3, for example, are utilized. The instruction store will be hypothetically partitioned as shown in Figure 2, and type 1 instructions would be located somewhere within the lower 7/8 of the addressing space in the I-store 2; type 2 instructions would be located in the upper 1/8 of the address spaces in I-store 2. This is accommodated prior to loading of the instructions into the I-store 2 from an external source through the ILR 8 as shown in Figure 1 when the instructions have been link edited and examined and have been assigned I-store addresses for loading in accordance with the type of instruction that they represent.

When a type 1 add instruction as shown in Figure 3 is to be exe- cuted, it is first fetched from the I-store 2 and placed into the IDR 1. At th same time, the three high order address bits of the location in I-store 2 from which the instruction was fetched will be loaded into the IDSR 10; they will be supplied thereto by the IAR 3. This is the fetching cycle, and when it is complete, the registers that constitute the IDSR 10 and the IDR 1 will contain the specific codes as shown in Figure 4. These have come from the segment identified by 010 in I-store 2 and have the arbitrary contents identified as 726380, the hexidecimal instruction of a type 1 add register's example as shown from Figure 3.

At this point, the contents of the IDR 1 and the IDSR 10 are taken together to be used as the source of an address for looking up the decode execution code represented by the instruction in the IDM 5. At the end of the decoding cycle, the execution register 4 EXR will be loaded with the decoded instruction codes that were found in the IDM 5 at the location specified by the content of the IDR 1 and the IDSR 10 and IDR 1 and IDSR 10 will be loaded with the next instruction to be decoded and the address in I-store from which it was fetched, respectively. The state of the registers at the start of the execution cycle is indicated in

Claims

1. A method of decoding properly for execution normally incompatible plural format instructions into compatible single format execution codes in a processor, characterized by the steps of:
fetching sequentially for decoding and execution said instructions from a first addressable storage means; and
decoding sequentially said fetched instructions for execution in said processor by accessing execution codes stored in a second addressable storage means by utilizing at least a portion of the address in said first addressable storage from which each said instruction was fetched together with at least a portion of said fetched instruction itself as the address for said second addressable storage means.

2. The method as described in Claim 1, further comprising the step of segregating by format said instructions into separately identifiable groups of locations in said first addressable storage means.

3. The method as described in Claim 1, further comprising the step of retaining sequentially the addresses in said first addressable storage means from which said process instructions are fetched.

4. The method as described in Claim 1 to 3, further comprising the step of arranging for access the execution codes for each said instruction in said second addressable storage means, said arranging being done at locations therein that are determined by at least a portion of each said instruction together with at least a portion of the address in said first addressable storage means wherein said process instruction resides.

5. A data processing system having plural and compatible format instruction and decoding execution capability, characterized in that it comprises:
a first addressable storage means acting as an instruction store, said instruction store containing instructions in two or more incompatible formats for execution in said data processing system, and
a second addressable storage means acting as an execution code store, the contents of said execution code store being arranged to be addressed for each said instruction in said instruction store by an address utilizing at least a portion of said instruction together with at least a portion of the address in said instruction store at which said instruction resides.

6. System as described in Claims 5 further comprising means for retrieving sequentially the addresses in said first addressable storage means from which said process instructions are fetched.

RA9-87-006

FIG. 1B

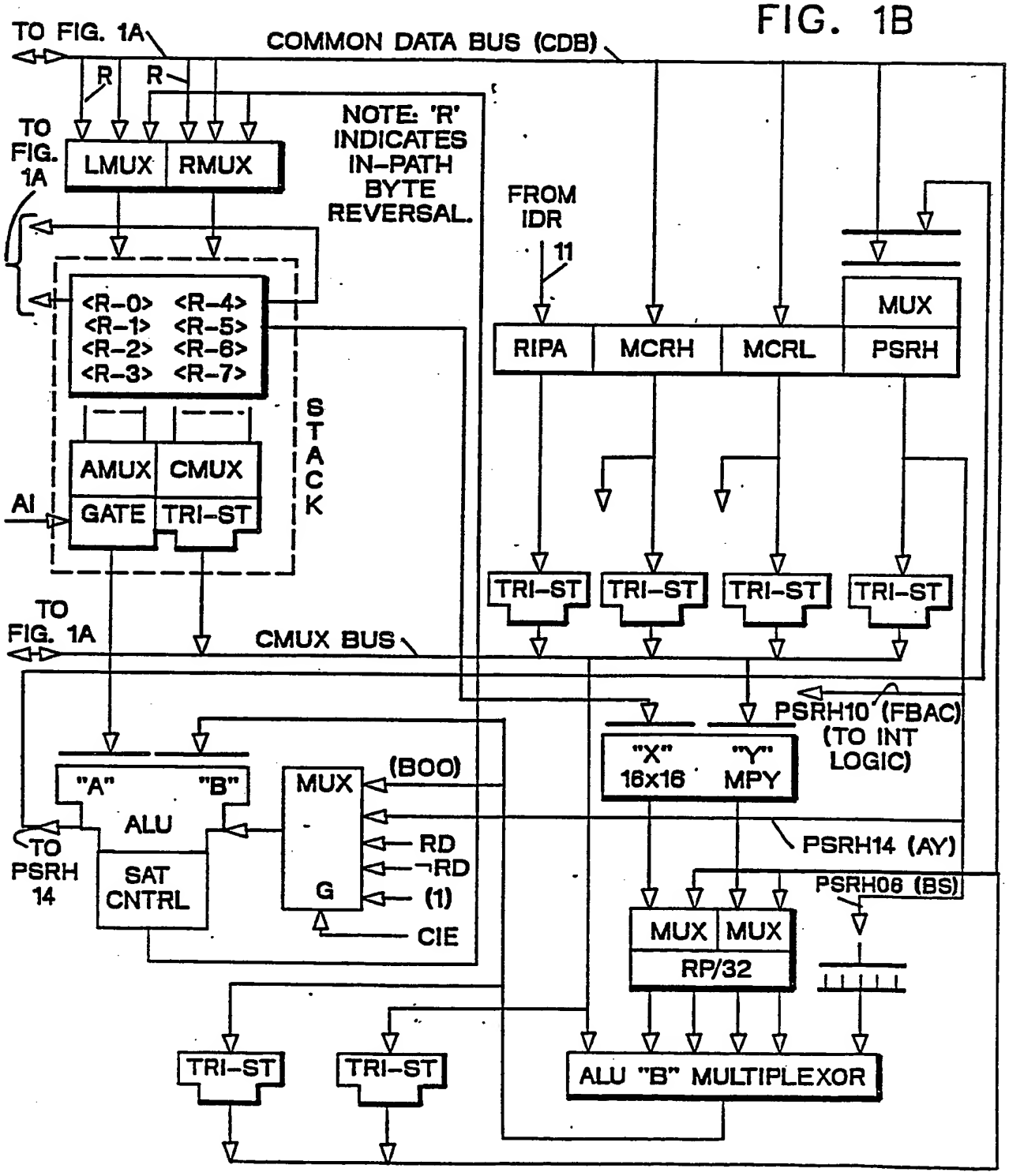
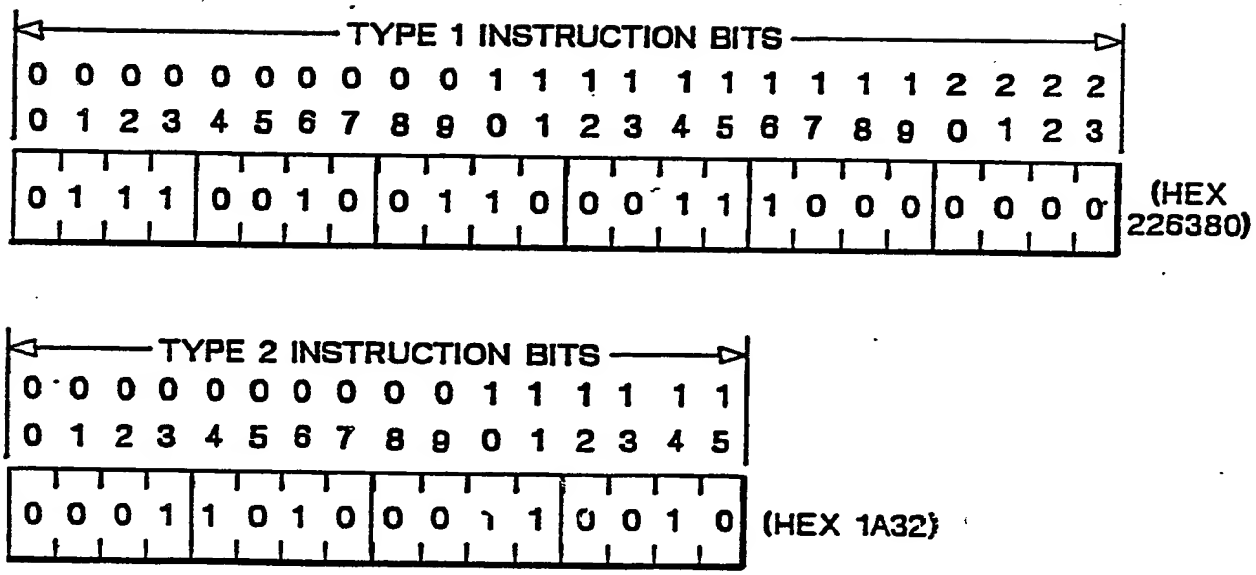


FIG. 3


TYPE 1 INSTRUCTION BITS
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
0 1 1 1 0 0 1 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0
(HEX 226380)
TYPE 2 INSTRUCTION BITS
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
0 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0
(HEX 1A32)

RA9-87-006

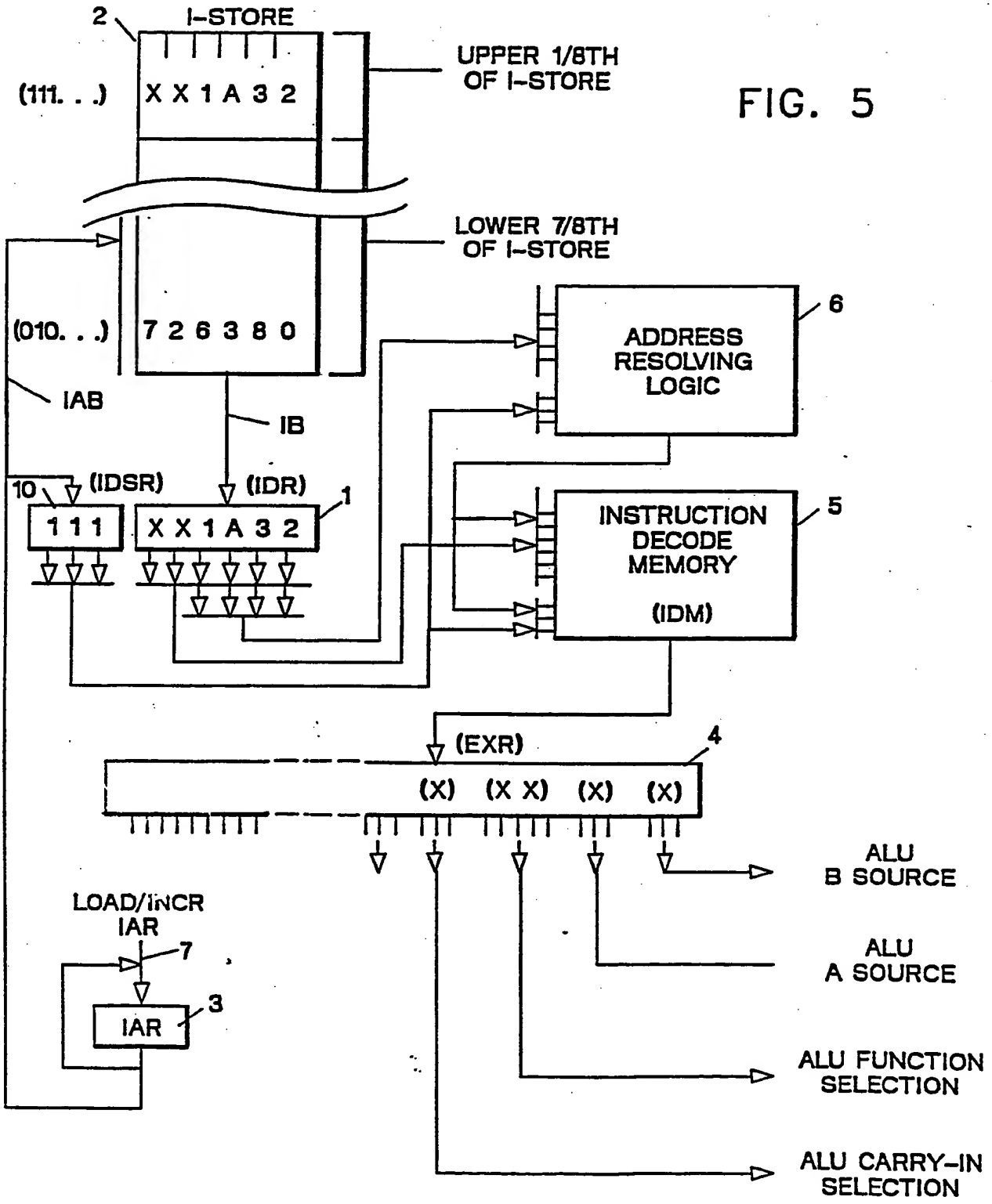


FIG. 5